IN THE CLAIMS

1. (currently amended) A method, comprising:

inserting an on-processor register allocation instruction within a machine code description of a routine if a function call instruction is found within said routine.

2. (original) The method of claim 1 further comprising configuring said allocation instruction to allocate only for the live information that exists within said routine when said inserted allocation instruction is executed.

3. (original) The method of claim 2 wherein said live information is determined by identifying information that is referred to before and after said function call.

4. (original) The method of claim 3 wherein said information identified after said function call extends to an exit block of said routine.

5. (original) The method of claim 4 wherein the worst case path to said exit block is allocated for.

6. (original) The method of claim 3 wherein said information identified after said function call extends to a post-dominator block of said routine.

7. (original) The method of claim 6 wherein the worst case path to said post-dominator block is allocated for.

8. (original) The method of claim 2 wherein said live information is information that is local to said routine.

9. (previously presented) The method of claim 8 wherein a processor said routine is to be executed upon has its associated register space partitioned into register space used only for local information and register space used only for global information, said allocation instruction pertaining only to said register space used for local information.

10. (original) The method of claim 2 wherein said live information includes global information.

11. (currently amended) The method of claim 1 wherein said allocation instruction is inserted just before a machine code representation of said function call.

12. (original) The method of claim 1 wherein said allocation instruction is inserted in a pre-dominator basic block of said function call.

13. (original) The method of claim 12 wherein said allocation instruction is inserted in said pre-dominator basic block of said function call if there exists a post-dominator basic block of said function call.

14. (currently amended) A method comprising:

~~inserting multiple allocation instructions within a routine by searching for one or more functional characteristics within said routine and~~ inserting ~~an~~ on-processor

register allocation instructions within ~~said~~ a machine code description of a routine ~~for each~~ because functional characteristics ~~that~~ selected from the group consisting of:

a) a loop that exists within a control flow graph of said routine;

b) a software pipelined loop; and,

c) a function call

~~that is~~ are discovered within said routine.

15. (currently amended) The method of claim 14 wherein at least one of said ~~one or more~~ discovered functional characteristics includes ~~a~~ said loop that exists within a control flow graph of said routine.

16. (previously presented) The method of claim 15 wherein the allocation instruction inserted for said loop is inserted above said loop in said control flow graph.

17. (original) The method of claim 16 wherein said allocation instruction allocates for a worst case path to an exit block of said routine.

18. (original) The method of claim 16 wherein said allocation instruction allocates for a worst case path to a post-dominator block of said routine.

19. (currently amended) The method of claim 14 wherein at least one of said ~~one or more~~ discovered functional characteristics includes ~~a~~ said software pipelined loop.

20. (original) The method of claim 19 wherein the allocation instruction inserted for said software pipelined loop is inserted above said loop in said control flow graph.

21. (previously presented) The method of claim 20 wherein said allocation instruction allocates for a worst case path to an exit block of said routine.

22. (original) The method of claim 21 wherein said allocation instruction allocates for a worst case path to a post-dominator block of said routine.

23. (previously presented) The method of claim 14 wherein said one or more functional characteristics include a function call.

24. (currently amended) The method of claim 14 further comprising determining the number of on-processor registers to be allocated for an allocation instruction after a functional characteristic is found.

25. (currently amended) The method of claim 24 wherein all ~~of said one or more~~ functional characteristics from said group within said routine are discovered before said determining is performed.

26. (original) The method of claim 24 wherein said determining is performed before a next functional characteristic is discovered.

27. (original) The method of claim 14 further comprising building an understanding of said routine's control flow graph before said searching is performed.

28. (currently amended) A method, comprising:

a) performing a first allocation for a first amount of on-processor register space at the entry block of a routine;

b) performing a second allocation for a second amount of on-processor register space for the live information within said routine at the time of said second allocation;

c) performing a function call to a second routine; and,

d) performing a third allocation for a third amount of on-processor register space at the entry block of said second routine, said third amount of allocated on-processor register space ~~and~~ overlapping with said first amount of allocated on-processor register space ~~having a common register~~.

29. (original) The method of claim 28 wherein said live information is determined by identifying information that is referred to before and after said function call.

30. (original) The method of claim 29 wherein said information identified after said function call extends to an exit block of said routine.

31. (original) The method of claim 30 wherein the worst case path to said exit block is allocated for.

32. (original) The method of claim 29 wherein said information identified after said function call extends to a post-dominator block of said routine.

33. (original) The method of claim 32 wherein the worst case path to said post-dominator block is allocated for.

34. (original) The method of claim 28 wherein said live information is information that is local to said routine.

35. (previously presented) The method of claim 34 wherein a processor said routine is to be executed upon has its associated register space partitioned into register space used only for local information and register space used only for global information, said allocation instruction pertaining only to said register space used for local information.

36. (original) The method of claim 28 wherein said live information includes global information.

37. (original) The method of claim 28 wherein said second allocation is performed just before said function call.

38. (original) The method of claim 28 wherein said second allocation is performed in a pre-dominator basic block of said function call.

39. (original) The method of claim 28 wherein said second allocation is performed in said pre-dominator basic block of said function call if there exists a post-dominator basic block of said function call.

40. (original) The method of claim 28 further comprising compiling said routine.

41. (currently amended) A machine readable medium having stored thereon sequences of instructions which are executable by a digital processing system, and which, when executed by the digital processing system, cause the system to perform a method comprising:

inserting an on-processor register allocation instruction within a machine code version of a routine if a function call instruction is found within said routine.

42. (original) The machine readable medium of claim 41 further comprising instructions which cause a processor that executes said routine to configure said allocation instruction to allocate only for the live information that exists within said routine when said inserted allocation instruction is executed.

43. (original) The machine readable medium of claim 42 wherein said live information is determined by identifying information that is referred to before and after said function call.

44. (original) The machine readable medium of claim 43 wherein said information identified after said function call extends to an exit block of said routine.

45. (original) The machine readable medium of claim 44 wherein the worst case path to said exit block is allocated for.

46. (original) The machine readable medium of claim 43 wherein said information identified after said function call extends to a post-dominator block of said routine.

47. (original) The machine readable medium of claim 46 wherein the worst case path to said post-dominator block is allocated for.

48. (original) The machine readable medium of claim 42 wherein said live information is information that is local to said routine.

49. (previously presented) The machine readable medium of claim 48 wherein said processor said routine is to be executed upon has its associated register space partitioned into register space used only for local information and register space used only for global information, said allocation instruction pertaining only to said register space used for local information.

50. (original) The machine readable medium of claim 42 wherein said live information includes global information.

51. (currently amended) The machine readable medium of claim 41 wherein said allocation instruction is inserted just before a machine code representation of said function call.

52. (original) The machine readable medium of claim 41 wherein said allocation instruction is inserted in a pre-dominator basic block of said function call.

53. (original) The machine readable medium of claim 52 wherein said allocation instruction is inserted in said pre-dominator basic block of said function call if there exists a post-dominator basic block of said function call.

54. (currently amended) A machine readable medium having stored thereon sequences of instructions which are executable by a digital processing system, and which, when executed by the digital processing system, cause the system to perform a method comprising:

~~inserting multiple allocation instructions within a routine by searching for one or more functional characteristics within said routine and~~ inserting ~~an~~ on-processor register allocation instructions within said machine code description of a routine ~~for each~~ because functional characteristics selected from the group consisting of:

a) a loop that exists within a control flow graph of said routine;

b) a software pipelined loop; and,

c) a function call

~~that is~~ are discovered within said routine.

55. (previously presented) The machine readable medium of claim 54 wherein at least one of said ~~one or more~~ discovered functional characteristics includes ~~a~~ said loop that exists within a control flow graph of said routine.

56. (previously presented) The machine readable medium of claim 55 wherein the allocation instruction inserted for said loop is inserted above said loop in said control flow graph.

57. (original) The machine readable medium of claim 56 wherein said allocation instruction allocates for a worst case path to an exit block of said routine.

58. (original) The machine readable medium of claim 56 wherein said allocation instruction allocates for a worst case path to a post-dominator block of said routine.

59. (currently amended) The machine readable medium of claim 54 wherein at least one of said ~~one or more~~ discovered functional characteristics includes ~~a~~ said software pipelined loop.

60. (previously presented) The machine readable medium of claim 59 wherein the allocation instruction inserted for said software pipelined loop is inserted above said loop in said control flow graph.

61. (original) The machine readable medium of claim 60 wherein said allocation instruction allocates for a worst case path to an exit block of said routine.

62. (original) The machine readable medium of claim 61 wherein said allocation instruction allocates for a worst case path to a post-dominator block of said routine.

63. (previously presented) The machine readable medium of claim 54 wherein said one or more functional characteristics include a function call.

64. (currently amended) The machine readable medium of claim 54 further comprising sequences of instructions which cause the system to determine the number of on-processor registers to be allocated for an allocation instruction after a functional characteristic is found.

65. (currently amended) The machine readable medium of claim 64 wherein ~~all of said one or more~~ functional characteristics from said group within said routine are discovered before said determining is performed.

66. (original) The machine readable medium of claim 64 wherein said determining is performed before a next functional characteristic is discovered.

67. (original) The machine readable medium of claim 54 further comprising sequences of instructions which cause the system to build an understanding of said routine's control flow graph before said searching is performed.

68. (currently amended) A machine readable medium having stored thereon sequences of instructions which are executable by a digital processing system, and which, when executed by the digital processing system, cause the system to perform a method, comprising:

a) performing a first allocation for a first amount of on-processor register space at the entry block of a routine;

b) performing a second allocation for a second amount of on-processor register space for the live information within said routine at the time of said second allocation;

c) performing a function call to a second routine; and,

d) performing a third allocation for a third amount of on-processor register space at the entry block of said second routine, said third amount of allocated on-processor register space ~~and~~ overlapping with said first amount of allocated on-processor register space ~~having a common register~~.

69. (previously presented) The machine readable medium of claim 68 wherein said live information is determined by identifying information that is referred to before and after said function call.

70. (previously presented) The machine readable medium of claim 69 wherein said information identified after said function call extends to an exit block of said routine.

71. (previously presented) The machine readable medium of claim 70 wherein the worst case path to said exit block is allocated for.

72. (previously presented) The machine readable medium of claim 69 wherein said information identified after said function call extends to a post-dominator block of said routine.

73. (previously presented) The machine readable medium of claim 72 wherein the worst case path to said post-dominator block is allocated for.

74. (previously presented) The machine readable medium of claim 68 wherein said live information is information that is local to said routine.

75. (previously presented) The machine readable medium of claim 74 wherein a processor said routine is to be executed upon has its associated register space partitioned into register space used only for local information and register space used only for global information, said allocation instruction pertaining only to said register space used for local information.

76. (previously presented) The machine readable medium of claim 68 wherein said live information includes global information.

77. (previously presented) The machine readable medium of claim 68 wherein said second allocation is performed just before said function call.

78. (previously presented) The machine readable medium of claim 68 wherein said second allocation is performed in a pre-dominator basic block of said function call.

79. (previously presented) The machine readable medium of claim 68 wherein said second allocation is performed in said pre-dominator basic block of said function call if there exists a post-dominator basic block of said function call.

80. (previously presented) The machine readable medium of claim 68 further comprising compiling said routine.

COMMENTS

The enclosed is responsive to the Examiner's Final Office Action mailed on July 17, 2003. At the time the Examiner mailed the Office Action claims 1 through 80 were pending. By way of the present response, pursuant to a Request for Continued Examination (RCE) as provided under 37 CFR 1.114 which has been filed herewith, the applicant has: 1) amended claims 1, 11, 14, 15, 19, 24, 25, 28, 41, 51, 54, 55, 59, 64, 65 and 68; and, 2) neither added nor canceled any claims. As such, claims 1 through 80 remain pending. The applicant respectfully requests reconsideration of the present response and the allowance of claims 1 through 80.

In the Office Action response mailed July 17, 2003 the Examiner rejected each of the Applicant's independent claims 1, 14, 28, 41, 54 and 68 under the theory that that the Scales reference (U.S. Patent No. 5,950,228) could be combined with the Orr reference (US Patent No. 5,748,963) to cover claim language in the present application that expressed **a causal relationship** between the presence of a function call and the insertion of an allocation instruction (See, with respect to claim 1, pg. 2 of the Examiner's Office Action mailed 07/17/03 ("Orr . . . teach[es] the determination of a function call"); with respect to claim 14, pg. 3 of the Examiner's Office Action ("[claim 14 is] rejected for the same reasons put forth in the rejection of claim 1"); with respect to claim 28, pg. 7 of the Examiner's Office Action ("Scales discloses allocation instructions, and Orr teaches the determination of a function call in a routine")).

The Applicant respectfully disputes this rationale as explained in more detail below.

The Scales Reference

The Scales reference is devoted to multi-processor computing systems in which clusters of multi-processor platforms share each other's memory space. For example, referring to Figure 2 of Scales, the "rightmost" multi-processor platform (having eight processors) 211 is enabled to use the memory space 212 of the "leftmost" multi-processor platform (having two processors) 211. A concern in multi-processor environments is data coherency. According to Scales, coherency "means that only one processor can modify any part of the data at a time". Scales, Col. lines 26-27.

Scales states "[t]he main problem [with prior art approaches] is that data coherency can only be provided at large (coarse) sized quantities because typical [off platform memory accesses] are 4K or 8K bytes" which, unfortunately, can be "inconsistent with the much smaller sized data units accessed by many processes, for example 32 or 64 bytes". Because of the large difference between the amount of data actually transferred to/from off-platform memory and the amount data actually needing such transferal, inefficiency results in the form of "increase[d] network traffic", "degrade[d] system performance" and the "potential for race conditions". See, Scales Col. 1, lines 61 – Col. 2, line 3.

As a solution, Scales proposes "to allow the unit of transfer between [multi-processor platforms] to vary depending on the size of the accessed data structures. Coherency for large data structures should allow for the transfer of large units of data so that the time to transfer data can be amortized. Coherency for smaller data structures should allow for the transfer of smaller units of data." See, Scales, Col. 2 lines 13 – 19. Thus Scales proposes improved efficiency of multi-processor systems by adjusting the size of transferals of data between multi-processor platforms.

A particular aspect of the teachings of Scales that has been cited by the Examiner against the Applicant's independent claims involves a particular implementation detail of Scales. Specifically, "units of data transfer having variable granularities" may be realized by "inserting special allocation instructions in the programs which explicitly define the [unit of data transfer] size". See, Scales, Col. 16, line 56 to Col. 17, line 12. Thus the particular implementation detail highlighted by the Examiner teaches, for purposes of customizing the size of units of data transfer between multi-processor platforms, inserting an allocation instruction if a program defines the unit of data transfer size.

That is, Scales teaches a **causal relationship** between the presence of a data size definition and the insertion of an allocation instruction.

The Orr Reference

The Orr reference is directed to virtual machine/interpreter based software technologies (e.g., Java, Pascal) in which a level of generic machine code

(referred to as "p-code" or "byte code") exists between the source code and the actual machine code. That is, a source code level description of a program is first compiled into "p-code" level instructions; then, the p-code level instructions are interpreted by an interpreter so as to produce sequences of machine code level instructions that are targeted for a specific processor. This is a significant contrast against those technologies in which machine code targeted for a specific processor can be obtained directly by compiling the source code. See, Orr, Col. 1, lines 11 – 31.

A particular feature of Orr that the Examiner has identified involves improving the speed at which function calls are made. The speed improvement is achieved by, at the p-code level, replacing parameters of a function call instruction with a reference to a specific "dictionary" entry where a memory address for the routine being called is found. Here, without the above described modification, the appearance of the function call instruction would cause the "dictionary" to be searched for the specific entry; which, in turn, wastes time.

A dictionary is a reference having an entry for each of a plurality of routines that may be called (e.g., akin to a lookup table). The dictionary correlates, for each routine having an entry, the number of arguments used by the routine and the routine's signature (i.e., its "identifier" (i.e., its name) and the types of input data that are to be passed to it) to a location in memory where the routine can be found. According to standard execution of a function call, the dictionary is searched for a "match" of signature and argument number; and, the memory address of the "matching" routine is "returned" as a response to the

function call instruction. The program then begins execution of the called routine from the provided address. The Orr invention effectively eliminates the need for searching the dictionary by directly replacing the identifier/name and argument number search parameters of the function call instruction with a reference to the specific dictionary reference containing the needed memory address. See, Orr Col. 1, line 65 through Col. 2, line 32; and, Col. 3, lines 1 – 21.

Therefore Orr teaches a **causal relationship** between the presence of a function call instruction and the replacement of function call parameters with a dictionary entry.

The Examiner's Arguments

Each of the Examiner's rejections are based on a combination of Scales and Orr under 35 USC 103. The Applicant believes each of these rejections to be in error as articulated in more detail immediately below. Section 2141 of the MPEP provides (emphasis added):

> "[w]hen applying 35 USC 103, the following tenets of patent law *must* be adhered to:
>
> (A) The claimed invention must be considered as a whole;
> (B) The references must be considered as a whole and *must suggest the desirability* and thus the obviousness of making the combination;
> (C) The references must be viewed without the use of impermissible hindsight vision afforded by the claimed invention; and
> (D) Reasonable expectation of success is the standard by which obviousness is determined." MPEP 2141.

With respect to the requirement that there must be some suggestion of the desirability of making a particular combination, the Applicant (based on the above described perspectives of the Scales and Orr references) does not understand the Examiner's reasoning.

Specifically, in order for the Examiner's combination to at least pass an initial threshold of sensibility there must be: 1) some suggestion that there is equivalence or sameness between an allocation instruction and the replacement of function call parameters; or, 2) some suggestion that there is equivalence or sameness between a function call instruction and the definition of a unit of data transfer size. The Applicant respectfully requests the Examiner to **identify** which relationship above is being used by the Examiner to support the combination of Scales and Orr.

That is, has the Examiner concluded that the motivation to modify Orr with the teachings of Scales lies in the scientific reasoning that faster function calls can be achieved with the insertion of an allocation instruction if a function call is found because dictionary searches will be avoided (i.e., there exists some suggestion that an allocation instruction will replace function call parameters)?

Or, has the Examiner concluded that the motivation to modify Scales with the teachings of Orr lies is the scientific reasoning that efficient data coherency can be achieved in a multi-processor environment through the insertion of an allocation instruction if a function call is found because variable sized data

transfers can be achieved (i.e., there exists some suggestion that a function call instruction defines the size of a unit of data transfer)?

Or, is the Examiner's scientific reasoning something other than those listed above? If so, could the Examiner please be reasonably specific with respect to the precise teachings of Scales and Orr and the exact nature of the motivation to combine (e.g., something more specific than a naked assertion of "faster operation" or "improved operation")?

The Applicant respectfully submits that any of the possible "scientific reasonings" that could be used in support of a combination of Scales and Orr that are directed at teaching the insertion of an allocation instruction (or the allocation of register space) because a function call exists (like those scientific theories listed just above) **will be absurd**. This is so because it is **absurd** to regard an allocation instruction in the same light as the replacement of function call parameters and it is **absurd** to regard a function call in the same light as a definition of data size.

If the Examiner disagrees the Applicant respectfully requests the Examiner to articulate a precise scientific theory that is not absurd, or, to correct the Applicant with respect to the teachings of Scales and Orr.

With the expectation that the Examiner will be unable to do so, the Applicant respectfully submits that the combination of Scales and Orr are impermissible because there exists no motivation to combine these references that is not absurd. Therefore each of the Applicant's independent claims are allowable. Therefore all of the Applicant's independent claims are allowable.

CONCLUSION

In light of the foregoing comments the Applicant respectfully submits that claims 1 through 80 are patentable and the Applicant respectfully requests the allowance of same.

Applicants respectfully submit the present application is in condition for allowance. If the Examiner believes a telephone conference would expedite or assist in the allowance of the present application, the Examiner is invited to call Robert O'Rourke at (408) 720-8300.

Authorization is hereby given to charge our Deposit Account No. 02-2666 for any charges that may be due.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN

Date: 10/15/03

Robert B. O'Rourke
Reg. No. 46,972

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025-1026
(408) 720-8300